IMPROVING TSS/360 PERFORMANCE BY TUNING THE TABLE-DRIVEN SCHEDULER

Jerry Kelcey Baird

United States
Naval Postgraduate School

THESIS

IMPROVING TSS/360 PERFORMANCE
BY TUNING THE TABLE-DRIVEN SCHEDULER

by

Jerry Kelcey Baird

Thesis Advisor: G.H. Syms

June 1971

Approved for public release; distribution unlimited.

Improving TSS/360 Performance By Tuning the Table-Driven Scheduler

by

Jerry Kelcey Baird
Captain, United States Marine Corps
B.S., University of Utah, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

During the period of time from August 1970 through January 1971 and while employing the TSS/360 Time-Sharing System at this institution, it was observed by the user community that the performance of the system was poor compared to the previously used time-sharing system - the CP/67 (version 3, from Cambridge Research Center). For this reason, the problem of improving TSS/360 performance was undertaken as a thesis project. Specifically, the improvements consist of an increase in system performance - responsiveness and throughput - by judiciously adjusting the parameters of the TSS/360 Table-Driven Scheduler in accordance with the Principles of Balanced-Core Time and Working Set Size.

A number of test runs were made, and the results are given, employing different schedule tables. A set of benchmark programs (or script) were developed and used with these tests that were characteristic of a "typical" or "realistic" load at this installation.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 7

II. NATURE OF THE PROBLEM ---------- 9

III. PRINCIPLES AND CONCEPTS ---------- 10

A. PERFORMANCE ---------- 10

B. FOLDING PROGRAMS ---------- 12

C. LOCALITY OF REFERENCE ---------- 13

D. WORKING SET AND WORKING SET SIZE ---------- 14

E. BALANCED CORE TIME ---------- 15

IV. TSS/360 TABLE-DRIVEN SCHEDULER ---------- 16

A. STRUCTURING OF SCHEDULE TABLE ENTRIES --------- 20

1. The Starting Set ---------- 21

2. The Looping Set ---------- 23

3. The AWAIT Set ---------- 24

4. The Holding Interlock Set ---------- 24

5. The Waiting-For-Interlock Set ---------- 25

V. EXPERIMENTAL PROCEDURE AND RESULTS ---------- 25

A. DEVELOPMENT OF A BENCHMARK ---------- 25

B. MEASUREMENT TECHNIQUES ---------- 27

C. TOOLS FOR MEASURING PERFORMANCE ---------- 27

D. PRESENTATION AND DISCUSSION OF RESULTS -------- 28

1. Test 1 ---------- 30

2. Test 2 ---------- 30

3. Test 3 ---------- 33

4. Test 4 ---------- 33
5. Test 5 ---------- 37
VI. CONCLUSIONS AND RECOMMENDATIONS ---------- 43
APPENDIX A SCHEDULE TABLE PARAMETER DEFINITIONS ------- 47
APPENDIX B SCHEDULE TABLES USED FOR DIFFERENT TESTS --- 54
LIST OF REFERENCES ---------- 60
INITIAL DISTRIBUTION LIST ---------- 62
FORM DD 1473 ---------- 63

LIST OF FIGURES

1. Contents of the Schedule Table Entry ---------- 17

2. Maintenance of Task Status Index Lists ---------- 19

3. TSS/360 Schedule Table Example ---------- 22

4. Naval Postgraduate School IBM 360 Model 67 ------- 29

5. TSS/360 Test 1 Load Conditions and Performance Statistics ---------- 31

6. TSS/360 Test 2 Load Conditions and Performance Statistics ---------- 32

7. TSS/360 Test 4 Load Conditions and Performance Statistics ---------- 34

8. TSS/360 Test 5 Load Conditions and Performance Statistics ---------- 39

9. Response Time Comparisons for Fortran Compilation of Test 5 ---------- 40

10. Response Time Comparisons for Small PL/1 Compilations of Test 5 ---------- 40

11. Response Time Comparisons for EDIT Script of Test 5 ---------- 41

12. Throughput Comparisons for Fortran Compilation of Test 5 ---------- 41

13. Throughput Comparisons for EDIT Script of Test 5 ---------- 42

14. Schedule Tables Used for Different Tests --------- 54

B1. Old TSS/360 Schedule Table ---------- 54

B2. New IBM Research Schedule Tables ---------- 56

B3. Test 4 (Run 5) Schedule Table Modifications - 58

B4. Test 5 - Schedule Table Modifications ------- 59

ACKNOWLEDGEMENTS

I am grateful to Professor Gordon Syms for his help as my thesis advisor. Without his consistent motivation to seek improvements in the performance of time-sharing systems, this particular thesis project would not have been possible.

I am also greatly appreciative of the efforts and many hours that Mrs. Pimporn C. Zeleny, of the W.R. Church Computer Center programming staff, spent in helping with the TSS/360 system. I would also like to thank my classmates who helped with the terminal sessions.

I. INTRODUCTION

Since the initial release of the time-shared operating system, TSS/360, in October 1967, performance has improved significantly with each subsequent release. However, for the period from August 1970 to February 1971, the Naval Postgraduate School converted from the CP/67 time-sharing system to the Time-Shared System, TSS/360, and found the new system undesirable to the user community in terms of system performance - responsiveness and throughput. Because of its poor performance, TSS/360 was short-lived at the school and was never given an opportunity through testing and evaluation procedures to indicate its worth and future use as a good performance time-sharing utility.

The objective of the research for this thesis was to find ways of improving the performance of TSS/360 at the Naval Postgraduate School.

After having read the available literature on the TSS/360 system, it seemed that the key area for study and work was the scheduling algorithm. At first a simulation model of the TSS/360 scheduling algorithm looked like a fruitful area of endeavor; however, this area was abandoned because of the time factor in building such a detailed simulation model and since it had taken John McCredie and Steven Schlesinger of Carnegie-Mellon [Ref.1] about a year to write such a model. They describe a modular simulation

model designed to aid in determining the value of entries in the TSS/360 schedule table. They showed that a useful model can be designed to answer a limited set of questions about a complex system without detailed modeling of all system components.

Another alternative, and the one that was finally pursued, investigated, and tested, was that of methodically altering the parameters of the TSS/360 Table-Driven Scheduler to achieve optimum system performance for the particular IBM 360/67 hardware configuration available.

In order to test and evaluate the performance of TSS/360, which was based on five test runs with different schedule tables, it was first necessary to construct a set of test programs (a benchmark or script) that would be representative of a realistic load on the system. This alone was a difficult task since, in a time-sharing environment, many user programs are in contention for similar system resources and at any particular time, there could be many demands or requests for a particular resource.

Another objective of this paper is to compile the available literature regarding the performance of time-sharing systems that apply to TSS/360 and show by experimental tests that these principles and concepts improve system performance.

II. NATURE OF THE PROBLEM

When this institution purchased the IBM 360/67 computing system in 1968, TSS/360 time-sharing operating system was not yet available (with the bugs removed), but the future employment and implementation of TSS was the big factor and sales promotion feature in purchasing the IBM 360/67 hardware configuration. As an alternate, CP/67 (version 3 from Cambridge Research Center) was used successfully for nearly two years. Then announcement was made to the user community that in August 1970 the IBM 360/67 would be operated as it was originally intended and that TSS/360 would replace the CP/67 time-sharing system. Prior to implementation by the computer facility programming staff, the TSS/360 was debugged and tested; however, little consideration was given to tuning the system to the job load of the Naval Postgraduate School environment.

As previously stated, the TSS/360 time-sharing system was used for about six months during which time the performance was quite unacceptable to the user community. It was observed that heavy paging users could ruin the performance; i.e., a few users manipulating large matrices or having many subroutines not properly linked could decrease the responsiveness to the other users. It was for this reason that this thesis project was initiated and motivated.

The two basic approaches that have been used for investigation of existing time-sharing systems have utilized either the analytic or simulation techniques. The analytic approach was the technique used to improve system performance of TSS/360. By methodically adjusting the parameters of the TSS/360 Table-Driven Scheduler using the principles of Balanced-Core Time and Working Set Size, improvement of the performance of the system can be achieved. Walter J. Doherty [Ref.2] showed that the performance of Release 4 Schedule Table of TSS/360 at the T.J. Watson Research Center was dramatically improved in a three-month period.

III. PRINCIPLES AND CONCEPTS

The principles and concepts discussed in this section are a compilation of the available literature regarding the improvement of performance of time-sharing systems as they pertain to TSS/360.

A. PERFORMANCE

Performance, appraised by Calingaert [Ref.3] as an independent entity, does not exist. The concept of performance can have a broad spectrum of meaning to different classes of people. However, fundamentally, performance of a computer is defined as the degree to which a computing system meets the expectations of the person involved with it. Some of the terms that are often included as aspects of performance are

responsiveness, throughput, turn-around time, availability, reliability, number of terminals supported, CPU utilization, channel and device utilization, and efficiency.

To a user of TSS/360 sitting at a terminal, the ability of the system to respond to his commands is his predominant view of performance [Ref.4]. A terminal user does not care if only one person or a hundred people are using the system simultaneously with him so long as the user thinks that there is a complete and dedicated computer at his disposal to provide certain services to him. A user would be much more irritated if he expected a TSS/360 edit request to respond in three seconds but it took five seconds than if he expected a response of ten minutes to some complex mathematical equation but it took thirty minutes. In other words, the system should be much more responsive to those requests to which a user expects an immediate reply, than to those requests during which the user knows that his attention can be turned elsewhere. (He could execute these programs in the background batch operation if the response is too slow.) This was a primary assumption that was made while setting out to improve TSS/360 performance.

It is most important to a system manager to know the number of terminals that TSS/360 can support, and it is also important to consider the categories of work that the terminal users are doing. As Doherty points out in his paper, an intuitively obvious but rarely mentioned concept is that for some categories of trivial work, the number of terminals

receiving adequate response may increase only after a threshold of human performance is reached. In other words, if the system is responding at a rate slower than a person's response time, any initial improvements in system performance will first result in the user's getting more work done; and only then will the system be able to handle more users at that level of responsiveness. By allowing longer delays in processing long-running programs as the load increases, it is possible to ensure that the very short jobs will constantly be provided with a fast response.

B. FOLDING PROGRAMS

Sayre [Ref.5] states: "By the unfolded form of a program we mean the form a program would take if it had available to it a large enough uniform memory to hold both itself and its data....On the folded forms the addresses have been rearranged -- folded-to-fit into the smaller address space actually available." In TSS/360, unfolded forms of programs and data exist in virtual memory. When a program is executed, portions of the program and its data are brought automatically into main memory for execution, which will result in automatic folding of the program if its complete execution space requirements are larger than the main memory available to hold it. McCredie [Ref.6] expressed in his paper that excessive overhead and long delays while pages are transferred into and out of core are two potential dangers of paging designs. It is important to fold a program into as small a

space as possible to prevent a degenerate situation called "thrashing" from occurring due to an unnatural folding. "Thrashing," as Denning [Ref.7] states, may also occur when a page is pushed from core to make room for another, but then is demanded again and brought back into core. Many programs can reach this state, and the paging rate can get so high that all productive work ceases. It is important to maintain a high degree of folding since it permits many programs to be folded into main core simultaneously, thereby providing a potentially significant increase in the level of multiprogramming. The dynamic relocation hardware available on the IBM 360/67 makes the automatic folding concept possible.

C. LOCALITY OF REFERENCE

The program performance on any paging system is directly related to its page demand characteristics. A program which behaves poorly accomplishes little computation on the CPU before making a reference to a page of its virtual memory that is on back-up storage, and thus it spends a good deal of time in waiting for pages to be read into core memory. A program which behaves well references storage in a more acceptable fashion, utilizing the CPU longer before referencing a page which must be brought in from back-up storage. This characteristic of storage referencing is often referred to as a program's locality of reference and can be found in Brawn's and Gustavson's paper [Ref.8]. Therefore, a program's locality of reference will influence the degree of folding to

which that program can be subjected with a minimal influence on its performance. Doherty has shown that a program with good locality will run more efficiently in a small execution space than one with poor locality.

D. WORKING SET AND WORKING SET SIZE

P.J. Denning [Refs.9 and 10] has investigated working set models with regard to program behavior in a virtual memory environment such as in the IBM 360 Model 67. The working set W(t,T) of a program is the set of pages referenced in the T page references immediately prior to time t. As time progresses, W(t,T) may or may not change; however, the better the program's locality of reference, the less likely it is that $W(t+1,T) \neq W(t,T)$. From Denning's paper, it appears natural to try to fold a program in such a way that the program's working set for a given time interval fits entirely in core memory. Reports of Fine, Jackson, and McIssac [Ref.11] provide some experimental evidence that the working set concept is a reasonable assumption for program paging behavior. Denning defines the working set size S(t,T) of a program, at time t, as the number of pages contained in the working set W(t,T). Therefore, it is possible to have the working set size remain unchanged and have the working set change. It appears natural to try to refold the program whenever its working set changes but, as Doherty indicates in his paper, it is difficult to do since it is not known in advance just when the working set is changing. So in most paging systems, a working set size change is more

easily detectable; hence, it is possible to detect working set changes at least when the working set size changes. Doherty describes a method for doing this, and his method is outlined below. The dynamic relocation hardware of the Model 67 system makes the application of this concept possible.

Using the concepts of working set, working set size, and locality of reference, Doherty states:

> "During a single interaction between a user at a terminal and TSS/360, several programs are usually executed for that user. Thus for the virtual execution time which spans this interaction, the working set size may or may not change; however, the working set will almost always change several times. Furthermore, for those programs having good locality of reference, the working set size during any one time slice will usually be much smaller than the working set size for the whole interaction time interval. And, in addition, the maximum working set size for all the time slices will probably always be smaller than the working set size for the whole interaction time interval. For those programs having poor locality of reference, the working set size for each time slice may frequently approach the working set size for the entire interaction time interval. Good locality relates more to the rate at which new pages enter W(t,T) than to its actual size."

E. BALANCED CORE TIME

From the previous discussion, programs having poor locality of reference and a large working set size would greatly reduce the level of multiprogramming if allowed to remain in core for very long periods of time. This result would affect throughput and responsiveness, since any new demands for service could not be honored quickly because core would be tied up. The Principle of Balanced-Core Time states that the length of the time slice in terms of virtual CPU execution time for any one task is inversely proportional to the

working set size in that interval. Therefore, this concept will allow good locality programs to progress very rapidly, whereas it will minimize the elapsed time that any large program (large working set size) can tie up core memory. In other words, a minimum time slice length will then be set for programs with large S(t,T) and poor locality to prevent paging overhead from dominating the system. In order to compensate for this compromise, the duration between large program time slices will be made much longer than the duration between time slices for smaller working set size programs. As a result, the level of multiprogramming and responsiveness will increase since more core is available more often. In addition, the degree of CPU utilization will increase.

IV. TSS/360 TABLE-DRIVEN SCHEDULER

The table-driven scheduler [Refs.12 and 13] is an algorithm which schedules and dispatches tasks within the multiprogrammed, time-shared environment. More specifically, the scheduler consists of a set of programs in the resident supervisor of TSS/360 used for scheduling, and consists of a static and resident table consisting of a variable number (256 maximum) of 28-byte entries. The 28-byte entries are called levels of the schedule table of Schedule Table Entries (STE). Each entry in any one level of the schedule table contains sufficient information to completely control

the execution of a task. The format of the schedule table entry is depicted in Figure 1.

1 BYTE	1 BYTE	2 BYTES	1 BYTE	1 BYTE	2 BYTES
LEVEL	PRI-ORITY	QUAN-TUM LENGTH	MAX QUANTA COUNT	MAX PAGES ALLOWED	MAX DISK I/O

1 BYTE	1 BYTE	2 BYTES	1 BYTE	1 BYTE	1 BYTE
SCAN THRESH-OLD	PULSE LEVEL	AWAIT EXT.	DELTA TO RUN	TIME SLICE END	MAX PAGES TSE

1 BYTE	1 BYTE	1 BYTE			1 BYTE
		1 BIT	1 BIT	1 BIT	
AWAIT LEVEL	TWAIT LEVEL	RE-COMPUTE FLAG	PRE-EMPT FLAG	STEAL REQUEST	MAX RELOC. PER QUANTUM

1 BYTE	1 BYTE	1 BYTE	1 BYTE	1 BYTE
HOLDING INTER-LOCK	LOW CORE HOLDING INTER-LOCK	WAITING ON INTER-LOCK	CONVER-SATIONAL WRITE ONLY	LOW CORE FORCE TSE

1 BYTE	1 BYTE	2 BYTES
UNUSED	NEXT STEAL LEVEL	DRUM SHARE

Figure 1. Contents of the Schedule Table Entry

Each task which enters the system has another table to describe itself to the system called the Task Status Index (TSI). Each TSI has a pointer to a level in the schedule table. Therefore, by changing the value of that pointer a task will be given a completely new set of scheduling parameters.

All TSI's in the system are chained together on one of two lists called the active and inactive lists. The active list has two logical subdivisions called the dispatchable and eligible lists. The dispatchable list consists of tasks occupying core storage and waiting for the CPU, and in most cases, whose Scheduled Start Time (SST) is less than the Master Clock (MC). When the SST of a task is less than the Master Clock, the task is said to be behind schedule. Tasks in the dispatchable lists are ordered according to their status as "execute bound" or "I/O bound." Those with heavy paging demands (I/O bound) are dispatched first.

The eligible list consists of tasks which are waiting for entry to the dispatchable list, i.e., which are ready to execute but have not yet been brought into main storage. These tasks are ordered by priority with the lowest priority number first on the list.

The inactive list consists of tasks waiting on long delay type stimuli, such as a terminal interrupt. These tasks, which are in AWAIT or TWAIT status, are incapable of continuing execution until a particular interruption occurs. Figure 2 depicts the movement of tasks among these three lists.

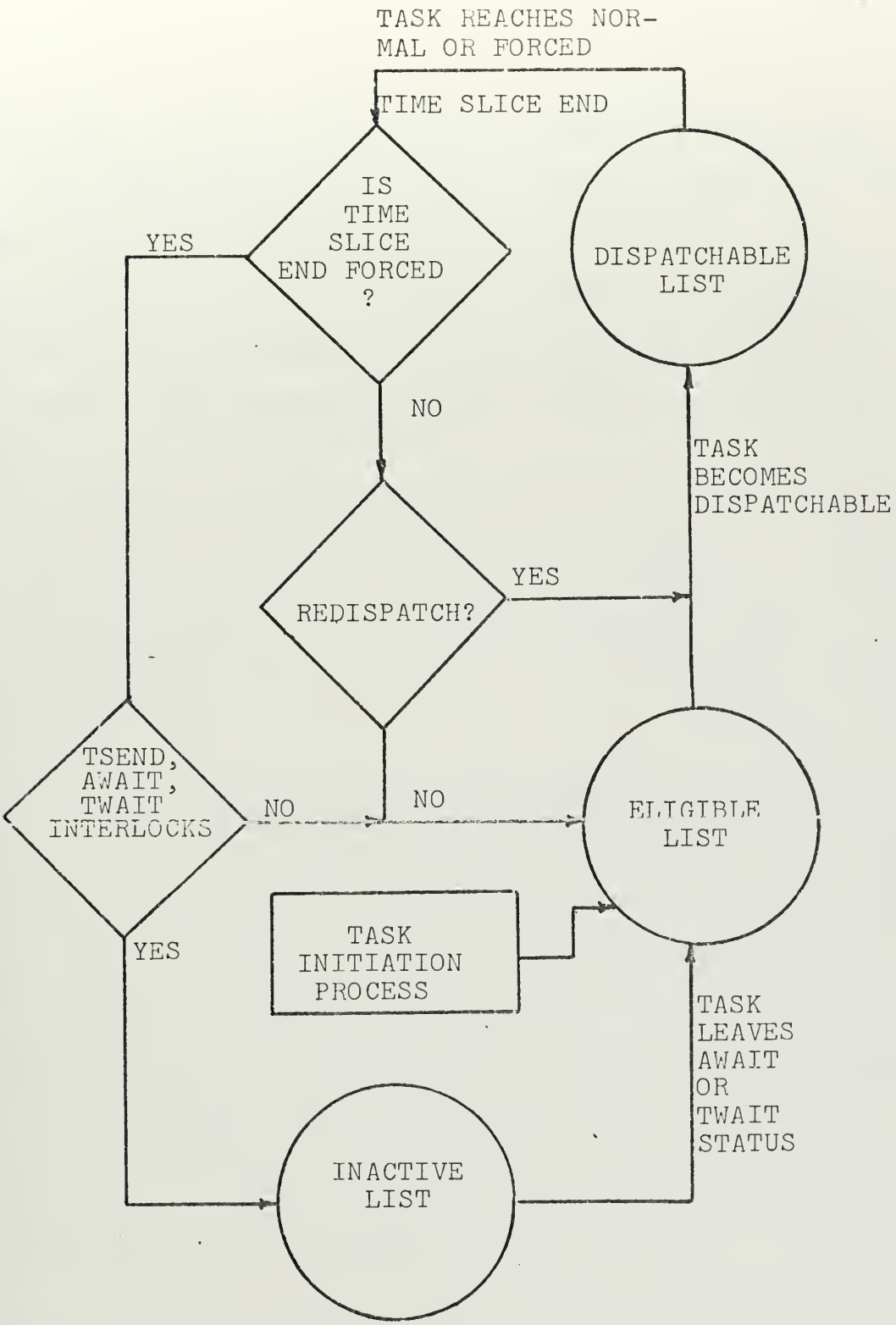

Figure 2. Maintenance of TSI Lists

The schedule table controls the order in which tasks are brought into the dispatchable list and the conditions under which the task will leave the dispatchable list.

The fields of each Schedule Table Entry (STE) can be classified into six logical areas.

The first is a set of fields that control dispatching, i.e., the order in which tasks move from the eligible to the dispatchable list (STEPRIOR, STEDELTA, STERCMP).

The second is a set of fields that provide limits that determine when a task shall be time sliced and leave the dispatchable list (STETSVAL, STEQUANT, STEMAXCR, STEAWTEX, STEPRMT).

Third is a set of fields that specify the level transition that will be made when the respective limit or stimulus has been reached (STEPULSE, STETSEND, STEMPRE, STETWAIT, STEAWAIT, ATEHLCK, STELCHL, STEWLCK, STECWO, STELCF, STEPRJ3, STENSL).

Next is one field which can stimulate a change in the order of tasks on the dispatchable list (STEMRQ).

Fifth is a set of fields which allow the resident supervisor to release some of a task's pages rather than time slice the task (STEST, STESRI).

Finally, there is a field which can override the system calculated drum share of private pages for a task (STEDSH).

Appendix A contains a description of each of the fields or parameters within a schedule table entry.

A. STRUCTURING OF SCHEDULE TABLE ENTRIES

By implementing the scheduling principles and concepts previously discussed, a wide spectrum of scheduling strategies can be implemented by altering only the entries within the schedule table.

In constructing the schedule tables according to the table scheduling strategies, different sets of levels are grouped according to some primary goals of scheduling. Several particular programs (tasks) are treated differently than other programs, e.g., system operator task, bulk I/O task, logon, and logoff. Figure 3 shows an example of a schedule table. All other programs are divided into the interactive and batch categories. In general, the same sets of levels exist for both kinds of programs, except that interactive programs have priority over batch programs; that is, interactive programs, initially, have a greater urgency to start than do the batch. The number of batch programs allowed to run simultaneously is arbitrarily restricted so that adequate space will be available for anticipated interactive programs. The interactive sets of table levels are grouped according to the following:

1. The Starting Set

The starting set of table levels are used to handle new inputs from the terminal. The functions of this set of table levels are to facilitate a rapid reply to the terminal, if possible, and to make an initial judgment of the present working set size of longer running programs, so that the best entrance to the looping set of table levels can be chosen for the particular program.

To accomplish this, several successive table levels with high priority, small execution time limits (100 milliseconds), and increasingly larger core space limits (16, 32,

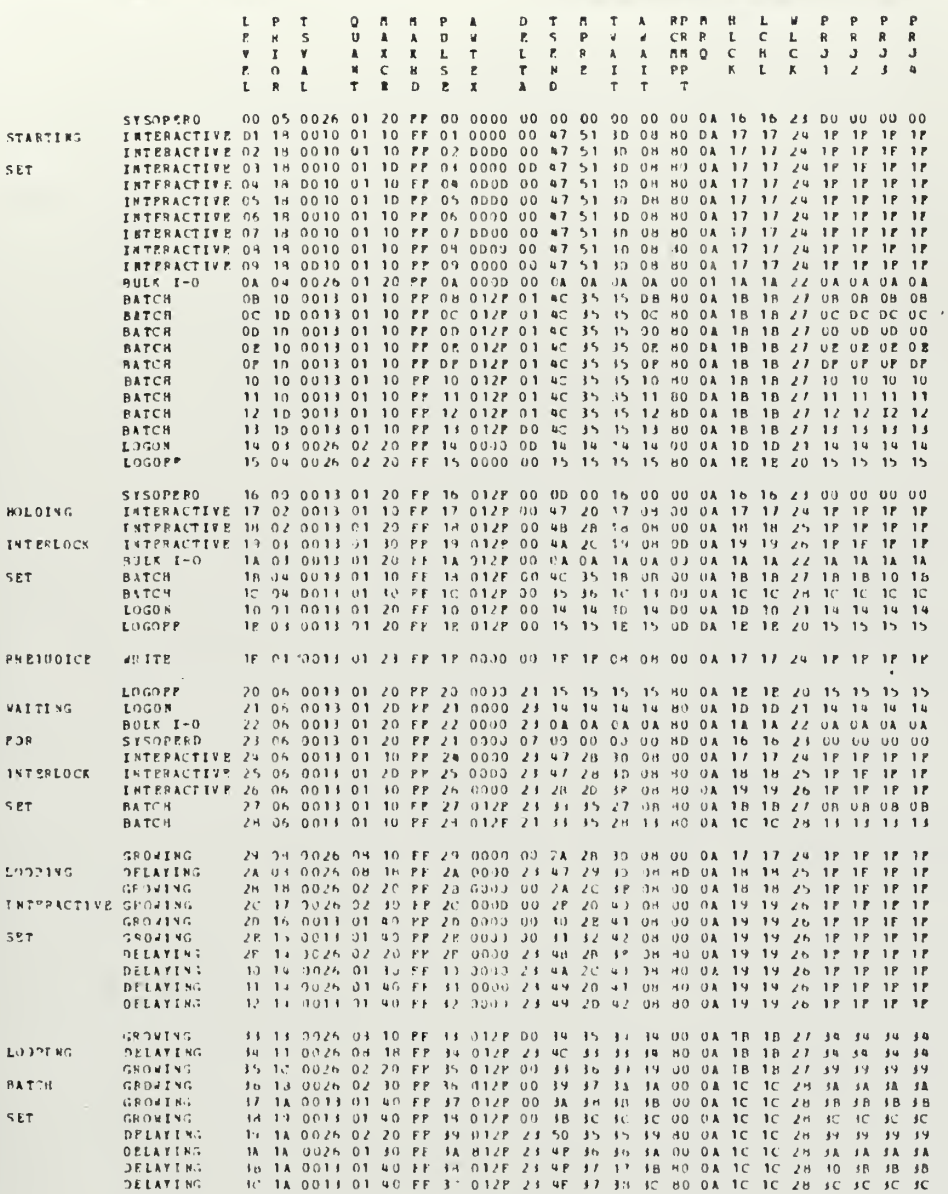

Figure 3. Schedule Table Example

48 pages) are established. As each program request enters from the terminal, it will move upward through these levels each time it exceeds its core space limit. Whenever the program exceeds its time limit at any of these levels, the core space limit of that level is used as the estimate of the program's present working set size. The program is then considered to be a longer running program and its future execution will be controlled by the looping set of table levels. Whenever a program exceeds its largest space limit, the largest allowable working set size (64 pages) will be used as the first estimate for future execution under control of the looping set.

When a program completes its execution, it is returned to the initial starting set table level to await the next input from the terminal.

2. The Looping Set

The looping set of table levels performs the following functions: they use the fields of the schedule table to follow a program's working set size by regularly overestimating and underestimating its time and core space requirements in a minimal fashion in accordance with the principle of balanced-core time; they cause the load that is generated by long running programs to be spread out in time to allow starting set entries to be processed rapidly; furthermore, they optimize the CPU utilization, and thereby penalize programs with poor paging characteristics by causing programs with minimal paging requirements to be selected to run much more

frequently than those with large paging requirements. This penalty occurs only when the program has poor locality of reference and a large working set size.

3. The AWAIT Set

The AWAIT set is a special set of table levels reserved for tasks doing tape I/O and other kinds of AWAIT operations. As previously described, in each table level there is an AWAIT extension field, which is an elapsed time interval during which a program's current working set pages are kept in core while the program remains idle in the AWAIT state. This can cause severe elongations of real time compared to virtual time; so that tasks with smaller values of virtual time are placed in this set of table levels rather than tasks of the same working set size which are in the looping set.

4. The Holding Interlock Set

The holding interlock set is also a special set that is reserved for programs that are currently holding interlocks on some system resource. (Holding an interlock means that some program is using a resource and preventing other programs from using that resource.) Programs in this set are given high priority so that the interlocked resource may be quickly released. An insignificant change in the working set size of programs operating in this set is assumed.

5. The Waiting-For-Interlock Set

The waiting-for-interlock set is another special set of levels for programs that are waiting for interlocks to be released that are currently being held by other programs in the holding interlock set. Until the interlock is released, programs in this set of table levels will not usually be considered for dispatching. An insignificant change in the working set size is also assumed for the interlock set.

V. EXPERIMENTAL PROCEDURE AND RESULTS

In order to make a number of test runs using different schedule tables, it was first necessary to provide a number of programs that would characterize a "realistic" load on the system relative to user demands at this school. This was necessary since TSS/360 was no longer the current time-sharing system in use at this computer installation, and a fixed load was needed to make valid performance comparisons.

A. DEVELOPMENT OF A BENCHMARK

As was previously discussed, the benchmark design concept for general purpose time-sharing systems is not an easy task to undertake and is confounded by two factors. The first is the variety of demands placed upon the system and second is the stochastic behavior of a time-shared system. Arnold D. Karush [Ref.14] presented an excellent discussion of the development of a benchmark design for the ADEPT Time-Sharing System at System Development Corporation, and pointed out

specific functional variables (compute activity, interactive activity, I/O activity, page activity, response allocation, user population, and swap activity) that affect system performance - specifically response time and throughput. Karush discusses two general program design techniques used to measure the performance of time-sharing systems - the analytical and stimulus methods. The analytical technique involves the insertion of probes into the system running under actual operating conditions. The stimulus technique consists of a "black box" concept and involves applying a controlled and measurable set of stimuli to the black box to activate the functional variables and then observe the effect of the stimuli upon the system.

The stimulus technique was used to develop the scripts for the experimental tests used in this paper; specifically a similar set of programs was used by the CP/67 and TSS/360 Time-Sharing System comparison group [Ref.15].

The final set of benchmark programs used in the test runs were as follows:

PLILG - large PL/I compilation

PLISM - small-sized PL/I compilation

FORT - Fortran program that is compiled

FORTEX - Fortran program that is executed

EDIT - execute routine that edits a simple program and files the edited program

PAGE - Fortran program which executes a large matrix multiplication.

B. MEASUREMENT TECHNIQUES

Two types of performance criteria were used to measure and judge the improvements in performance. The measurement consisted of observing the response times and throughput. The benchmark programs used in the tests were written to give the real time at the commencement and at the completion of a compilation. The throughput was calculated by observing the completed compilation or execution of a particular type job. The figure obtained by this procedure is called the throughput factor and was obtained as follows:

$$TP_i = SS/(RD \times NT_i)$$

where SS = Sample Size (number of completed jobs)

RD = Run Duration

NT_i = Number of terminals running program type i

In essence, the throughput factor is the reciprocal of the time to execute the program, modified by the size of the sample.

C. TOOLS FOR MEASURING PERFORMANCE

Unfortunately no hardware or software measurement device was available to measure resource utilization and performance of TSS/360 in this research. A software measurement tool called SIPE was obtained from IBM, but the required data analysis programs could not be obtained. Thus the actual measurements could be made, but there was no means of converting them into meaningful information on resource utilization.

The problem of developing a data analysis program to analyze the data from SIPE was considered as beyond the scope of this research.

D. PRESENTATION AND DISCUSSION OF RESULTS

Five test runs were conducted using different schedule tables. The results of these tests will be presented and discussed in this section.

The IBM 360 Model 67 configuration of the Naval Postgraduate School is shown in Figure 4 and is very similar, but not identical, to the IBM T.J. Watson Research Center's Model 67 configuration which Doherty used for his work. It should be noted that when the TSS/360 Time-Sharing System was implemented at this school for the months previously mentioned, the new IBM Watson Research Table by Doherty was not used. The initial schedule table used in TSS/360 is shown in Appendix B (Figure B1). This table provided poor performance to the user community. Just prior to TSS/360 being replaced by the new CP/67 version time-sharing system, the new IBM Research Schedule Table arrived and was implemented by extending and using important parameters that were never used in the old table. A significant improvement in performance was observed. This improved schedule table is shown in Appendix B (Figure B2). In fact about a fifty percent increase in utilization was observed, and yet, it was clear that more improvement could be obtained. It was not until these tests were begun that the new IBM Research Table (Figure B2 of Appendix B) was implemented and tested:

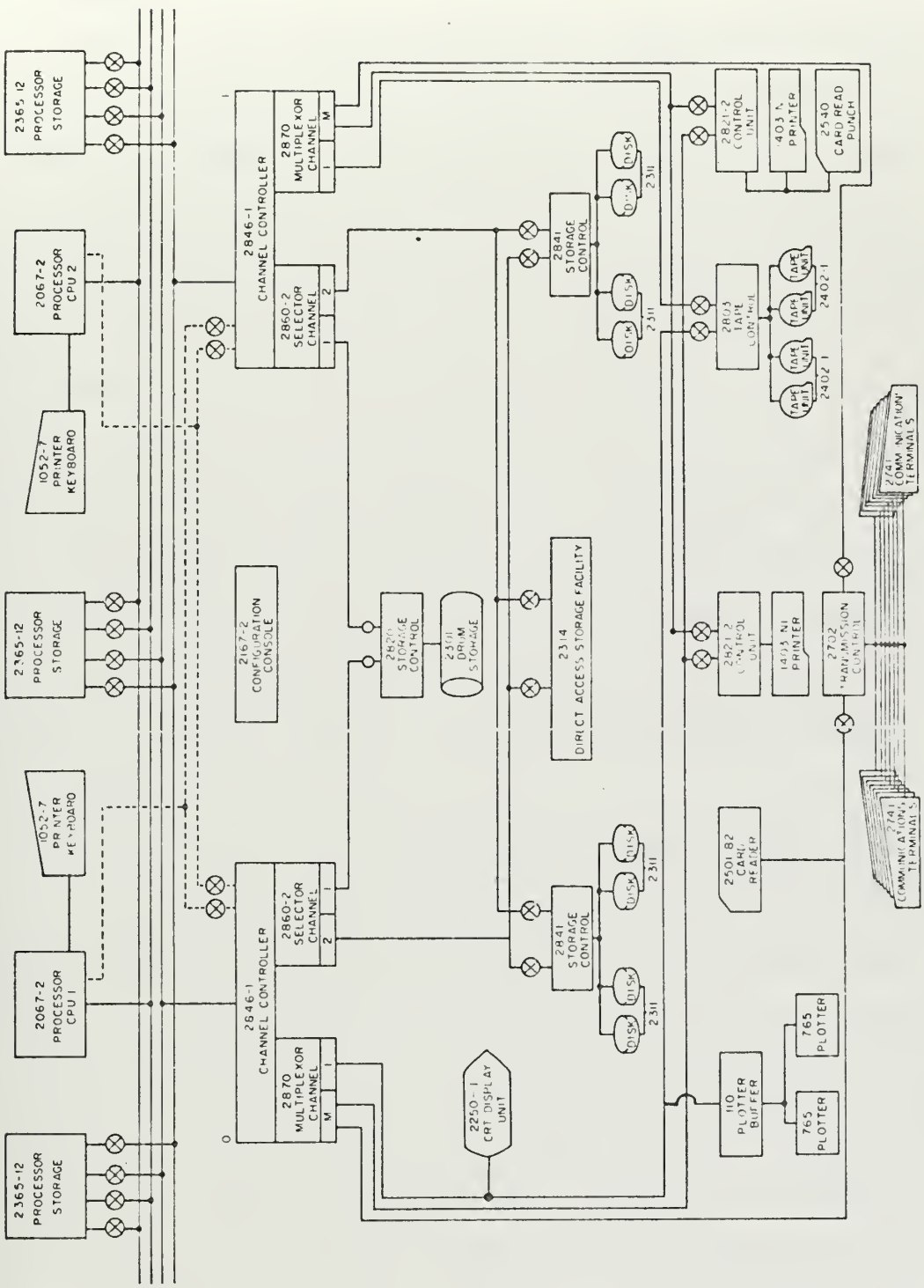

Figure 4. Naval Postgraduate School
IBM 366 Model 67

1. Test 1

Test 1 was a preliminary test in which the benchmark programs (or scripts) were initially used and in which the new IBM Watson Research Schedule Table (Figure B2 of Appendix B) was used. The load configuration and performance statistics for this test can be seen in Figure 5. Run six, operating with a good sampling of all the script except paging, produced a mean response of 8 min 37 sec for a large PL/1 compilation, 4 min 30 sec for a small PL/1 compilation, 1 min 8 sec for a Fortran compilation and 48 sec for an edit. This test did not provide a heavy load to the system. This table, however, did provide better responses than were previously observed by the user community when TSS/360 was running on a regular basis using the old schedule table.

2. Test 2

Test 2 was conducted, with the same schedule table used in Test 1, to provide a more realistic mix with different ratios of edit-to-run (compile and execute) programs and heavier load on the system. An important factor to remember in scheduling is that almost any scheduling technique will show similar results under light loads, but it is only when the demand for system resources gets large that scheduling differences are clearly indicated. The run durations were also lengthened to provide a steadier load on the system. The load characteristics and the performance statistics for test 2 are shown in Figure 6. Under this change in load, the response times have correspondingly increased significantly.

LOAD CONFIGURATION/DATA							
RUN NUMBER	1	2	3	4	5	6	7
PLILG (BIG PL/1)	1	1	1	1	1	2	1
PLISM (SMALL PL/1)	0	0	1	0	0	1	0
FORT (FORTRAN)	1	2	1	2	3	1	3
EDIT (EDIT SEQ.	8	10	12	12	14	14	16
PAGE	--	--	--	--	--	--	--
FORTEX (CPU BOUND)	2	2	3	3	3	3	4
RUN DURATION	16:53	13:29	14:52	17:18	14:32	16:48	20:23

RESPONSE TIME/THROUGHPUT STATISTICS								
PLILG	MEAN	1:51	3:32	4:16	4:26	7:54	8:37	9:11
	S DEV	0:07	0:21	0:16	0:13	0:11	0:54	0:51
	SS	8	4	4	3	2	4	2
	TP	0.47	0.30	0.27	0.17	0.14	0.12	0.10
PLISM	MEAN	--	--	3:11	--	--	4:30	--
	S DEV	--	--	0:28	--	--	0:58	--
	SS	--	--	4	--	--	4	--
	TP	--	--	0.27	--	--	0.24	--
FORT	MEAN	0:18	0:36	0:43	0:46	1:14	1:08	1:28
	S DEV	0:03	0:08	0:09	0:09	0:22	0:17	0:21
	SS	49	41	18	40	22	14	36
	TP	2.90	1.52	1.21	1.16	0.50	0.83	0.59
EDIT RE-SPONSE TO 6 EDIT COM-MANDS	MEAN	0:30	0:50	0:50	0:44	1:09	0:48	1:12
	S DEV	0:02	0:07	0:05	0:04	0:10	0:06	0:11
	SS	8	10	12	12	14	14	16
	TP	.059	.077	.067	.059	.071	.06	.05

Figure 5. TSS/360 Test 1
Load Conditions and Performance Statistics

LOAD CONFIGURATION/DATA					
RUN NUMBER		1	2	3	4
PLILG (BIG PL/1)		4	4	4	3
PLISM (SMALL PL/1)		3	6	5	5
FORT (FORTRAN)		7	7	7	5
EDIT (EDIT SEQ.)		4	4	4	8
PAGE		-	-	2	2
FORTEX (CPU BOUND)		6	3	2	1
RUN DURATION		17:32	15:42	37:45	43:20
RESPONSE TIME/THROUGHPUT STATISTICS					
PLILG	MEAN	21:58	21:48	35:07	31:07
	S DEV	1:25	3:26	0:30	3:39
	SS	4	4	4	3
	TP	.0570	.0635	.0262	.0230
PLISM	MEAN	15:38	17:30	27:03	22:02
	S DEV	1:42	2:34	1:31	3:22
	SS	4	6	8	6
	TP	.0760	.0636	.0423	.0276
FORT	MEAN	3:12	3:47	6:16	5:12
	S DEV	:39	1:03	2:00	1:10
	SS	38	26	36	36
	TP	.3100	.2371	.1361	.1660
EDIT RE-SPONSE TO 6 EDIT COM-MANDS	MEAN	3:59	4:56	11:47	9:26
	S DEV	-	-	-	-
	SS	8	5	8	16
	TP	.0761	.1060	.1055	.3690

Figure 6. TSS/360 Test 2
Load Conditions and Performance Statistics

Each test run was conducted under a terminal load of 27 users. Runs three and four were conducted with heavy paging, and as a result, a greater delay was observed in the response to a request. It was believed initially from the first two runs that the PL/I compiler characteristics produced the heavy load and the poor response, but when several heavy paging programs were added to the load, the performance was degraded even more. Paging in TSS/360 is handled by disk as well as drum, and since disk paging is slow, this might be one of the major problems.

3. Test 3

When test 3 was performed, one of the three core boxes failed. The results of this test indicate that TSS/360 operating with only two core boxes rather than three will produce a much lower system performance, so low that the results are meaningless for a comparison and are not included in this thesis.

4. Test 4

Without changing the schedule table of test 2, runs one through four were conducted to see if a different load would change the performance characteristics. Run four seemed to be a good sampling of the scripts and provided a heavy paging load, and the performance characteristics were about the same as that in run three of test 2. The load conditions and performance statistics for run four are shown in Figure 7.

LOAD CONFIGURATION/DATA						
RUN NUMBER		1	2	3	4	5
PLILG (BIG PL/1)		2	2	2	2	2
PLISM (SMALL PL/1)		3	3	3	3	3
FORT (FORTRAN)		5	5	5	5	5
EDIT (EDIT SEQ.)		12	10	8	6	8
PAGE		0	2	4	6	4
FORTEX (CPU BOUND)		2	2	2	2	2
RUN DURAITION		22:00	27:53	29:07	30:12	48:00
RESPONSE TIME/THROUGHPUT STATISTICS						
PLILG	MEAN	19:26	17:25	25:12	25:15	>46 min
	S DEV	0:26	0:09	1:38	1:48	-------
	SS	2	2	2	2	none finished (2)
	TP	*	*	*	*	*
PLISM	MEAN	12:58	17:14	18:48	20:44	11:22
	S DEV	1:01	3:02	0:50	1:40	1:27
	SS	*	*	*	*	*
	TP	0.291	0.251	0.220	0.192	0.350
FORT	MEAN	12:58	17:14	18:48	20:44	11:22
	S DEV	0:46	0:48	0:59	0:52	0:52
	SS	3	3	3	3	9
	TP	*	*	*	*	*
EDIT RESPONSE TO 6 EDIT COMMANDS	MEAN	3:00	4:08	5:25	6:12	8:38
	S DEV	0:19	0:26	0:32	0:26	2:39
	SS	24	22	15	10	18
	TP	0.091	0.079	0.064	0.055	0.0469

Note: * insufficient statistics.

Figure 7. TSS/360 Test 4
Load Conditions and Performance Statistics

Run five was conducted with the IBM Research Schedule Table patch altered. This modified IBM schedule table is shown in Appendix B (Figure B3). The table parameters that were altered for this run are found in the table levels of the Looping Interactive Sets and the Starting Set of the schedule table, since these sets provide areas in which the most improvements in performance could be realized. Several fields of the schedule table levels were altered, but none were changed drastically. This was done so that any degradation to the system which may have occurred from changing parameter values could be observed. The fields altered and the reasons for the alterations were as follows:

The delta-to-run parameters were increased so that the larger working set size programs could get into core faster but less frequently and remain there longer with larger values of time-slice end. The smaller size programs still get priority through the system.

The AWAIT extension field increases the time allowed for the larger size programs to remain on the dispatchable list before being forced to time slice. Since a task in AWAIT status is normally moved from the list of dispatchable tasks, and since this can cause a delay in redispatching the task, the idea was to make the AWAIT extension large enough to allow for completion of I/O operations.

A few priority values were changed, since these priorities determine the position a task will assume within

the list of eligible tasks; that is, low priority numbers are given precedence over higher priority numbers.

The Quantum Count and Quantum length fields were altered. These parameters determine the time slice, which is dynamic, for tasks assigned to this entry. Time slice duration equals Quanta Count times Quantum length x 3.33 milliseconds. These fields were altered to see the effect of the Balanced-Core Time Principle — where the time slice duration in terms of CPU execution time for a task is inversely proportional to the working set size in that time interval. This will minimize elapsed time that any large job can clog memory and allows jobs with good locality to progress rapidly.

The maximum core page residency values (MAXCR) have been selected to minimize task performance. Trivial and many non-trivial commands require less than 35(23 hexadecimal) pages allowed in the small conversational levels. However, some non-trivial commands take more pages, causing the task to move to other levels. If tasks with the Steal Request Flag (SRF) on move into core faster than pages can be released, they will exceed the MAXCR limit and be time sliced.

The maximum relocations per quantum field was altered. The smaller the value, the greater the guarantee the task will be considered I/O bound and its order in the dispatchable list will not change. Therefore, tasks which must be serviced can remain on or near the top of the

dispatchable list by assuming them to levels with small MRQ values.

The recompute flag field was altered. If tasks in these levels fall behind schedule, they will be given preference through the computation of their schedule start time. If the preempt flag is on, a task can be time slice ended if a higher priority task is ready and can not be dispatched.

The scan threshold fields were reduced in value, since it was felt that a 100% page stealing value was not necessary. The scan threshold is related to page stealing. It should be noted that the stealing mechanism which sets the steal flag was not implemented in the old schedule table that was used initially with the system. This field value was altered to allow page stealing.

As shown in Figure 7, by primarily increasing the delta-to-run and quantum fields, the large working size programs (PLILG) were penalized in their response times, whereas an improvement in response was observed in small PL/I and Fortran compilations. However, in the EDIT programs, response times were even worse during this run than before and the throughput factor went down.

5. Test 5

The last test was conducted using three different schedule tables. The characteristics for this test are shown in Figure 8. Unfortunately, there were only 20 terminals loading the system, since the other terminals were

inaccessible or inoperable. The time was also limited for these test runs so that the durations were shorter than was desirable. The schedule table for run three is shown in Figure B4 of Appendix B. The parameters that were altered for the schedule table for run three were the delta-to-run fields, which were set to very large values, and the quantum fields. Although the load was not as heavy as that of test 4, these test runs do show significant improvement, and the increased performance is the result of judiciously altering these parameters. The response times for PLILG programs for runs two and three were about the same, while the response time for FORT programs was better for run one than for two and three. However, it is expected that if a heavier load had been placed on the system, run three would have provided the better performance statistics. The response times for small size PL/I programs, and EDIT programs for run three show better response statistics, and the response time for FORT programs for run three shows an increase over run two. Figures 9,10 and 11 show the difference in response times for each of these runs. The throughput factor could not be obtained for big and small PL/I programs, but run three shows an increase in throughput over run two for FORT programs but about the same as run one. For EDIT programs, run three shows an increase in throughput over run one and two. Figures 12 and 13 show the difference in throughput.

LOAD CONFIGURATION/DATA				
RUN NUMBER		1 APP B FigB2	2 APP B Fig B3	3 APP B Fig B4
PLILG (BIG PL/I)		1	1	1
PLISM (SMALL PL/I)		2	2	2
FORT (FORTRAN)		5	5	5
EDIT (EDIT SEQ.)		4	4	4
PAGE		3	3	3
FORTEX (CPU BOUND)		2	2	2
RUN DURATION		44:08	25:43	17:48
RESPONSE TIME/THROUGHPUT STATISTICS				
PLILG	MEAN	44:06	20:50	20:50
	S DEV	-----	-----	-----
	SS	1	1	1
	TP	*	*	*
PLISM	MEAN	32:08	12:48	9:31
	S DEV	2:51	:11	:27
	SS	2	2	2
	TP	*	*	*
FORT	MEAN	:54	1:46	1:28
	S DEV	:15	:31	:02
	SS	93	30	38
	TP	.62	.3	.60
EDIT RE-SPONSE TO 6 EDIT COM-MANDS	MEAN	6:26	8:36	6:13
	S DEV	:07	:04	:06
	SS	22	9	8
	TP	.14	.11	.21

Note: * insufficient statistics

Figure 8. TSS/360 Test 5
Load Conditions and Performance Statistics

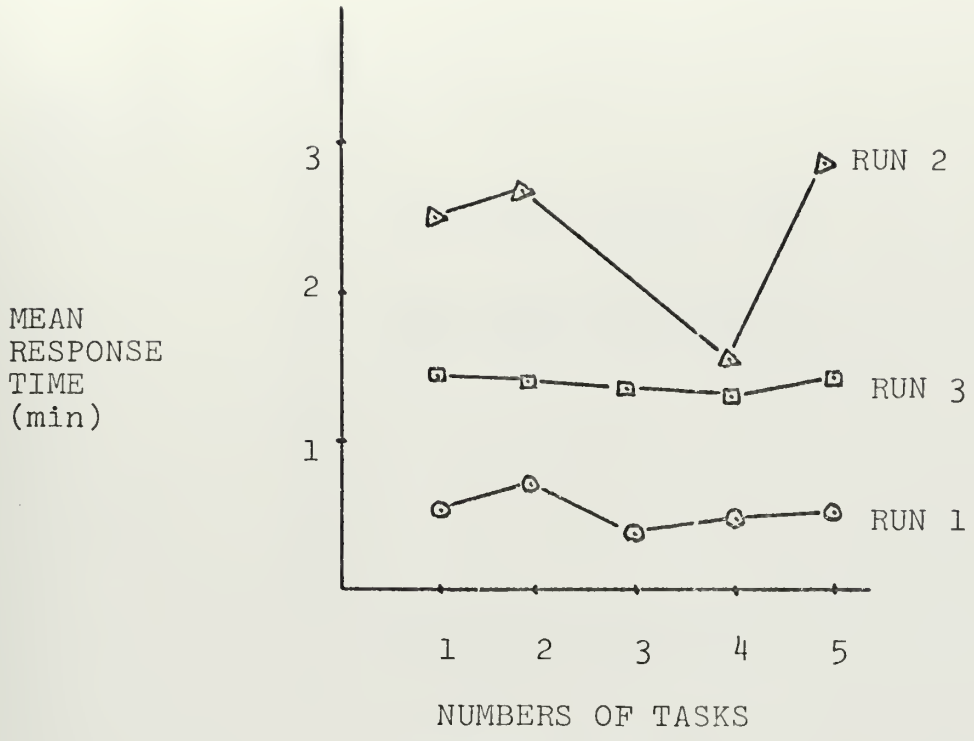

Figure 9. Response Time Comparisons for Fortran Compilations of Test 5

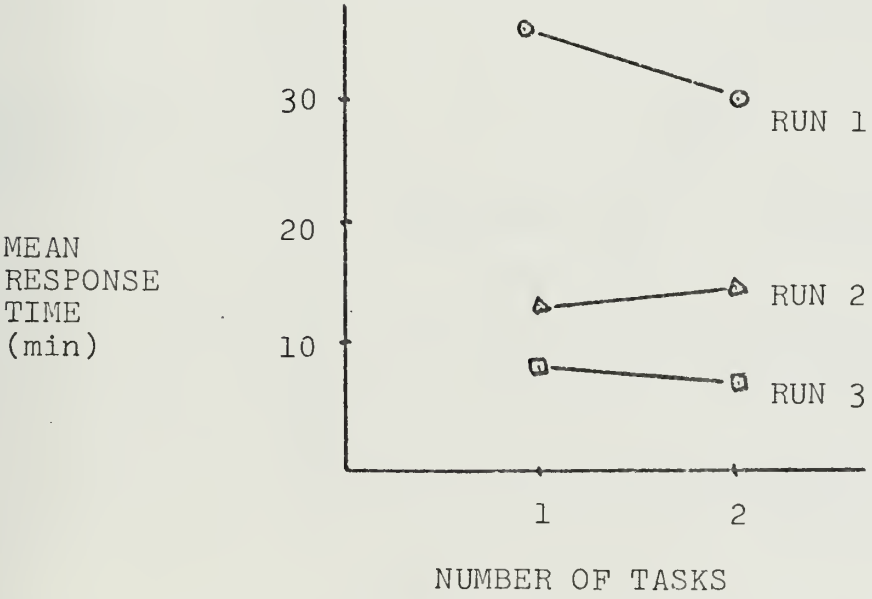

Figure 10. Response Time Comparisons for Small PL/I Compilations of Test 5

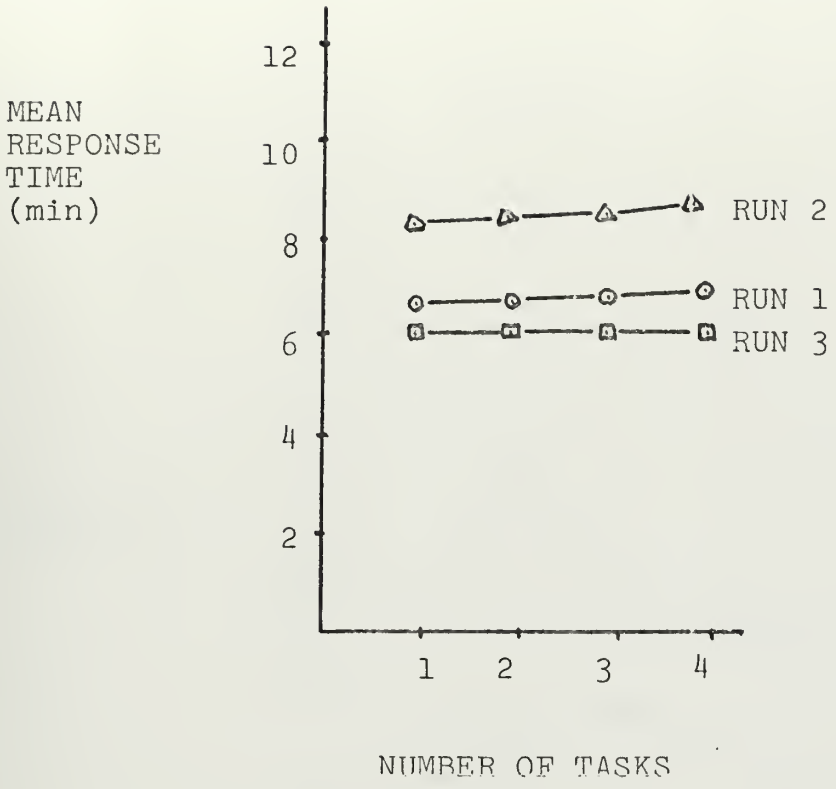

Figure 11. Response Time Comparisons for EDIT Script of Test 5 (6 EDIT Commands)

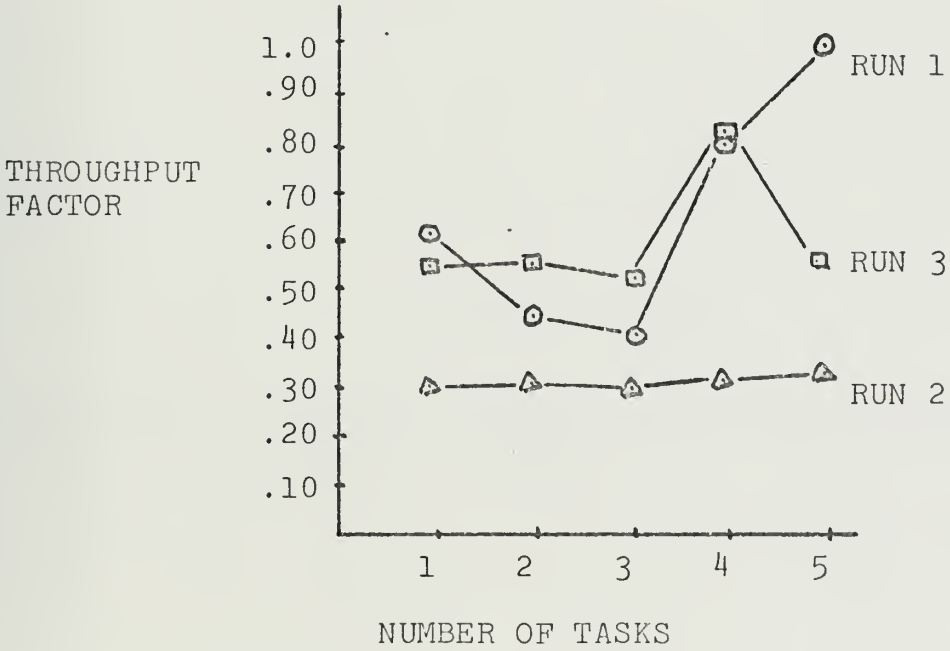

Figure 12. Throughput Comparisons for Fortran Compilations of Test 5.

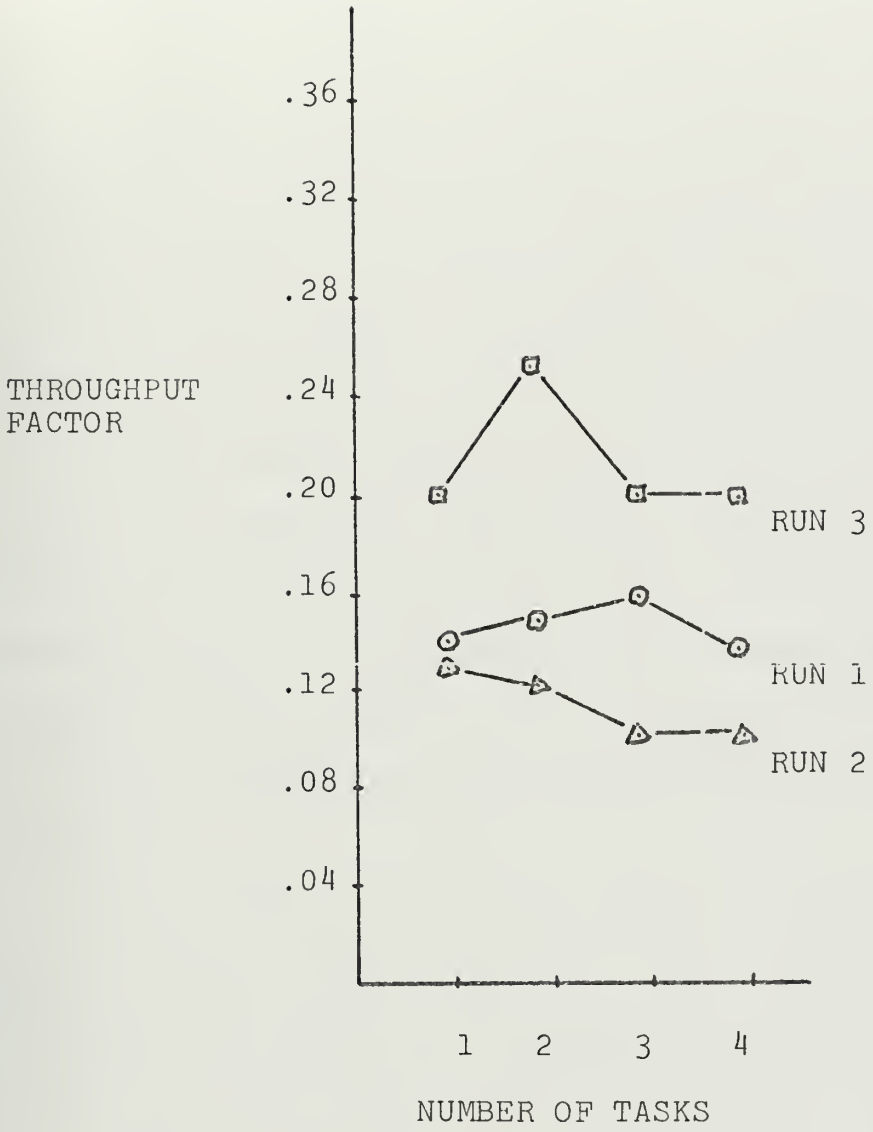

Figure 13. Throughput Comparisons for EDIT Script (6 EDIT Commands) of Test 5

VI. CONCLUSIONS AND RECOMMENDATIONS

The objectives of this paper were to organize all available literature regarding improvement of performance measures and techniques for the TSS/360 Time-Sharing System Schedule Tables and to show that these principles and concepts could be substantiated by performing experimental tests on the computer. As a result of altering the parameters of the TSS/360 schedule table, improved performance over the initial system performance, when the TSS/360 system was in full operation, was observed. From these tests it is evident that because of differences in the user community and in hardware configurations it is necessary that certain parameters in the table-driven scheduler be set for each installation to improve its system performance and thus maintain a satisfied user community.

It has been shown by these tests that the Naval Postgraduate School's Model 67 computer could support about 20-25 simultaneous users using a modified IBM Research schedule table, while maintaining a fair response to the trivial requests, and simultaneously servicing large users rather well. With more work on the schedule tables, better service could be provided for a greater simultaneous load. Once the TSS/360 Time-Sharing System was removed as the installation's time-sharing system, the time available for testing in this project was restricted. Many more valuable tests remain to be

performed to eventually optimize the performance of the system through the judicious alteration of the parameters of the TSS/360 table-driven scheduler.

There were many fields of the TSS/360 schedule table that were not varied and tested. For example, during the last test a table was designed to test the page drum mechanism, but since this mechanism was not yet implemented into the software, this table could not be employed. The present values of the schedule table at this installation show a 0000 default to the system calculated minimum number of pages on disk for all users. This value could be increased to allow some tasks to be allocated greater space on drum in order that fewer of their pages have to be moved from drum to disk. Nielson [Ref.16] in his simulation studies of time-sharing systems, showed that disk paging can be very slow and can reduce system performance substantially, and proposes that a drum be used in place of the disk. Since this installation used both drum and disk paging, an alternative solution could be to purchase another drum for paging. Also, revision of the disk management algorithm could be made.

As mentioned at the outset of this paper, a more flexible approach to evaluating the effects of changing different schedule table parameters on the performance of the system would have been the simulation approach rather than an analytical approach. However, such a simulation model would have to be limited in terms of expensiveness of design and running

time. Also, there is always the very difficult problem of validating the simulation model.

From the tests conducted in this paper, attempting to optimally tune the scheduler by trying various schedule tables in the proper type of environment is not an easy process. There were many factors which limited more speedy progress in tuning the scheduler to the job load of this school's environment. The benchmark that was implemented for the tests may not have accurately represented the user community, although a great effort in this direction was made. Since loads are constantly changing, it is important to develop a methodology for automatically producing scripts that are characteristic at this installation and then to verify that they are accurate.

The use of the TSS/360 software measurement technique, SIPE [Ref.17], would have been very valuable and helpful in establishing a good benchmark for developing, evaluating, and improving the interactive system. SIPE and its data-reduction program could also have been very helpful in evaluating changes to the schedule tables and the effects on system performance. These measurement tools could also provide valuable statistics about each task as it is being processed by the system. Software counters, as Doherty used, could also provide information about each task as it migrates through various levels of the schedule table to more accurately verify the principles of working set size and balanced core time. De Meis and Weizer [Ref.18] established by experimental means in developing RCA's Time Sharing Operating

System (TSOS) that by using certain measurement devices, the working set size and balanced-core time of programs can be monitored and verified.

Although SIPE produces some degradation to the system, this is not considered serious. The only way to monitor a system without altering its operation is by external hardware monitors. Schulman [Ref.19], for example, discusses a hardware monitor (SPAR) that also is used to measure TSS/360 and that does not degrade that system. Another tool that has been extremely useful in TSS/360 evaluation and improvement of performance is the instruction-time trace monitor (ITM) [Ref.20] which is a combination of software and hardware. With the aid of these additional measuring devices, it is believed that many more improvements could be made to the performance of TSS/360 by further adjustment of the entries in the schedule table.

APPENDIX A

SCHEDULE TABLE PARAMETER DEFINITIONS

LEVEL (STELEVEL), 1 BYTE

Relative entry number in schedule table. The level number is used to relative address within the schedule table.

PRIORITY (STEPRIOR), 1 BYTE

The priority of a level in conjunction with the Schedule Start Time (SST) is used to govern the allocation of CPU resources to a task. Only those tasks brought into the dispatchable list can increase in core usage. Zero is the highest priority. When seeking to bring a task into the dispatchable list, the highest priority task behind schedule is chosen. If no tasks are behind schedule, the highest priority task is chosen.

QUANTUM LENGTH (STETSVAL), 2 BYTES

The quantum length is the number of time units (one quantum) a task will be dispatched or the amount of time to be used as a factor in determining how long a task will be allowed to run before time-slice end. One unit represents 3.33 milliseconds. A quantum represents the maximum virtual memory time that a task will be dispatched. The system will then make a decision as to whether the task may have more CPU time based on the number of quanta used (see STEQUANT) or interrupted by a time-slice end.

MAXIMUM NUMBER OF QUANTA (STEQUANT), 1 BYTE

This field represents the maximum number of quanta (STES-VAL) a task may use or receive when it is in execution before a time-slice must occur.

MAXIMUM PAGES (STEMAXCR), 1 BYTE

This field represents the maximum number of private physical pages allowed in core before a time-slice end or page steal will occur. (see SCAN THRESHOLD)

MAXIMUM DISK I/O OR PAGE READS (STEMAXRD), 2 BYTES

This field represents the maximum disk reads or writes, or maximum number of page relocations a task will be allowed before a time-slice end will occur.

SCAN THRESHOLD (STEST), 1 BYTE

If the steal request flag (STESRF) is on, the resident supervisor will release some of a task's pages when the page count equals STEMAXCR (maximum core page residency values). The scan threshold is the percentage of STEMAXCR pages to be retained. The scan threshold is a percentage specified in hexidecimal (i.e., 80% = 80 base 10 = 50 base 16). When stealing occurs, the task is not time-sliced, but stays in the dispatchable list. However, the schedule table entry in the TSI is changed to the value specified in STENSL (next steal level).

PULSE LEVEL (STEPULSE), 1 BYTE

This field represents the schedule table level entry to be used if the pulse service is requested by the user. The pulse service allows the user to request a level change.

AWAIT EXTENSION (STEAWTEC), 2 BYTES

This field represents the maximum time that a task, issuing an AWAIT service, is allowed to remain in the dispatchable list while waiting for an I/O operation to be completed. The units are 3.33 milliseconds. If the I/O operation has not completed before the time limit specified, the task is time-sliced.

DELTA-TO-RUN TIME (STEDELTA), 1 BYTE

Specifies the real time interval at which a task is to be given a slice of CPU time. This field specifies a factor which is used to calculate a new Schedule Start Time (SST) for a task as it moves from one state to another; i.e., as the task becomes ready, in AWAIT or in TWAIT. The value in this field is multiplied by 852.5 milliseconds and may be combined with the master clock time or the old Scheduled Start Time if the old SST is negative to determine the task's new SST. If delta-to-run equals zero, the SST is set to zero and the task is automatically placed behind schedule. (see RECOMPUTE FLAG)

(TSE) TIME-SLICE END (STETSEND), 1 BYTE

This field represents the schedule table level entry to be used when a time-slice end occurs because of the maximum number of quanta (STEQUANT) or a maximum disk I/O (STEMAXRD) has been reached.

MAXIMUM PAGES TSE (STEMPRE), 1 BYTE

This field represents the schedule table level entry to be used when a tmme-slice end occurs because of the maximum pages in ccre (STEMAXCR) has been reached.

TWAIT TSE (STETWAIT), 1 BYTE

This field represents the schedule table level to be used after a time-slice end occurs because the TWAIT service has been used.

AWAIT TSE (STEAWAIT), 1 BYTE

This field represents the schedule table level entry to be used after a time-slice end occurs because the AWAIT service has been used.

RECOMPUTE FLAG (STERCMP), 1 BIT

If the recompute flag is on, the task's Scheduled Start Time is computed to place the task back on schedule as described above under delta-to-run (STEDELTA). If the flag is off, past performance (if behind schedule) is taken into account by calculating SST as the present time plus delta-to-run minus the amount behind schedule on the previous time-slice. NOTE: When a task enters the eligible list

directly from the dispatchable list, the schedule start time is calculated as if the recompute flag is off.

PRE-EMPT FLAG (STEPRMPT), 1 BIT

A task on the dispatchable list whose pre-empt flag is on may be forced to time-slice end so as to make room for a task from the eligible list having a higher priority.

STEAL REQUEST FLAG (STESRI), 1 BIT

A task on the dispatchable list whose steal request flag is on will have pages released (stolen) when its private pages in core reach the STEMAXCR limit. If pages are brought in faster than they can be released so that the STEMAXCR limit is exceeded, the task will be time-sliced.

MAXIMUM PAGE RELOCATIONS PER QUANTUM (STEMRQ), 1 BYTE

Specifies the maximum number of page relocation interruptions allowed per quanta before the task is declared paging bound; i.e., a task is considered to be execute bound if its number of page relocations per quantum is less than or equal to STEMRQ. Execute bound tasks are placed at the end of the dispatchable list to allow non execute bound tasks to overlap their paging I/O with execute bound tasks.

HOLDING INTERLOCK CHANGE LEVEL (STEHLCK), 1 BYTE

This field represents the schedule table level entry to be used when a time-slice end occurs (except for AWAIT or TWAIT) and the task is holding a Virtual Access Method (VAM) interlock.

LOW-CORE HOLDING INTERLOCK (STELCHL), 1 BYTE

This field represents the schedule table level entry to be used when a time-slice end occurs because of low-core and the task is holding a Virtual Access Method (VAM) interlock.

WAITING ON INTERLOCK CHANGE LEVEL (STEWLCK), 1 BYTE

This field represents the schedule table level entry to be used when a time-slice end occurs and the task is waiting on an interlock.

CONVERSATIONAL WRITE ONLY (STECWO), 1 BYTE

This field represents the schedule table entry to be used when a write without response message is sent to the terminal. The level change occurs without a time-slice end.

LOW CORE FORCED TIME-SLICE END (STELCF), 1 BYTE

This field represents the schedule table entry to be used when a task is forced to time-slice end for low-core and it is not holding an interlock.

PREJUDICE CATEGORY 3 (STEPRJ3), 1 BYTE

This field is not used in the system.

NEXT STEAL LEVEL (STENSL), 1 BYTE

This field represents the schedule table entry to be used when stealing occurs. The task is not time-sliced.

DRUM SHARE (STEDSH), 2 BYTES

This is the number of drum pages reserved for a task. There are about 500 pages available after startup on a one

drum system and 1400 pages on a two drum system. In general, the number of a task's private pages on drum is a function of the number of tasks logged on, the number of drums, and the time since the last time-slice. If the number of unassigned drum pages falls below a pre-determined limit, some pages are moved from drum to disk. Each task receives a system calculated minimum drum space. The drum share field allows some tasks to keep a large drum share. A value of zero defaults to the system calculated minimum.

APPENDIX B

SCHEDULE TABLES USED FOR DIFFERENT TESTS

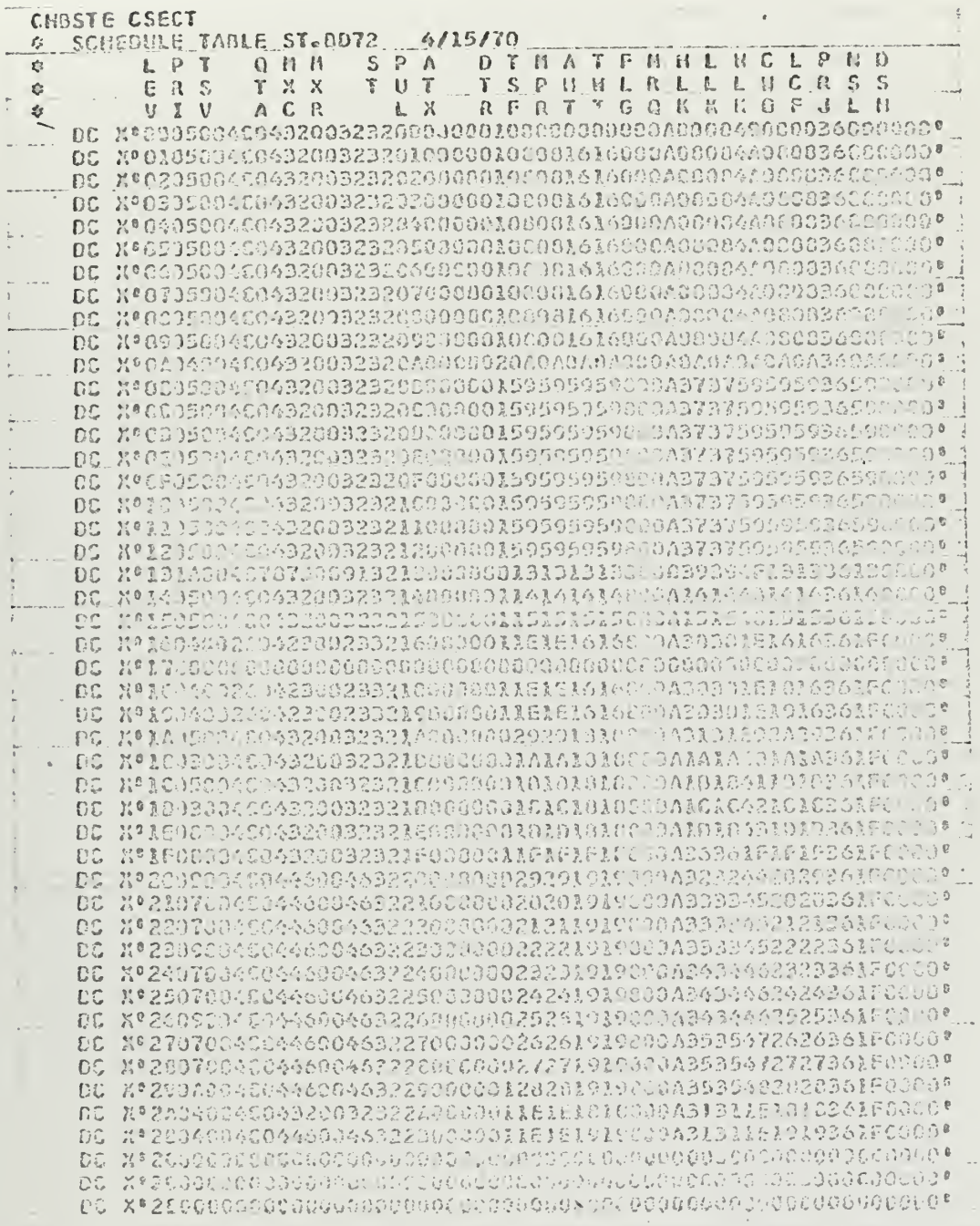

Figure 14 B1. Old TSS/360 Schedule Table

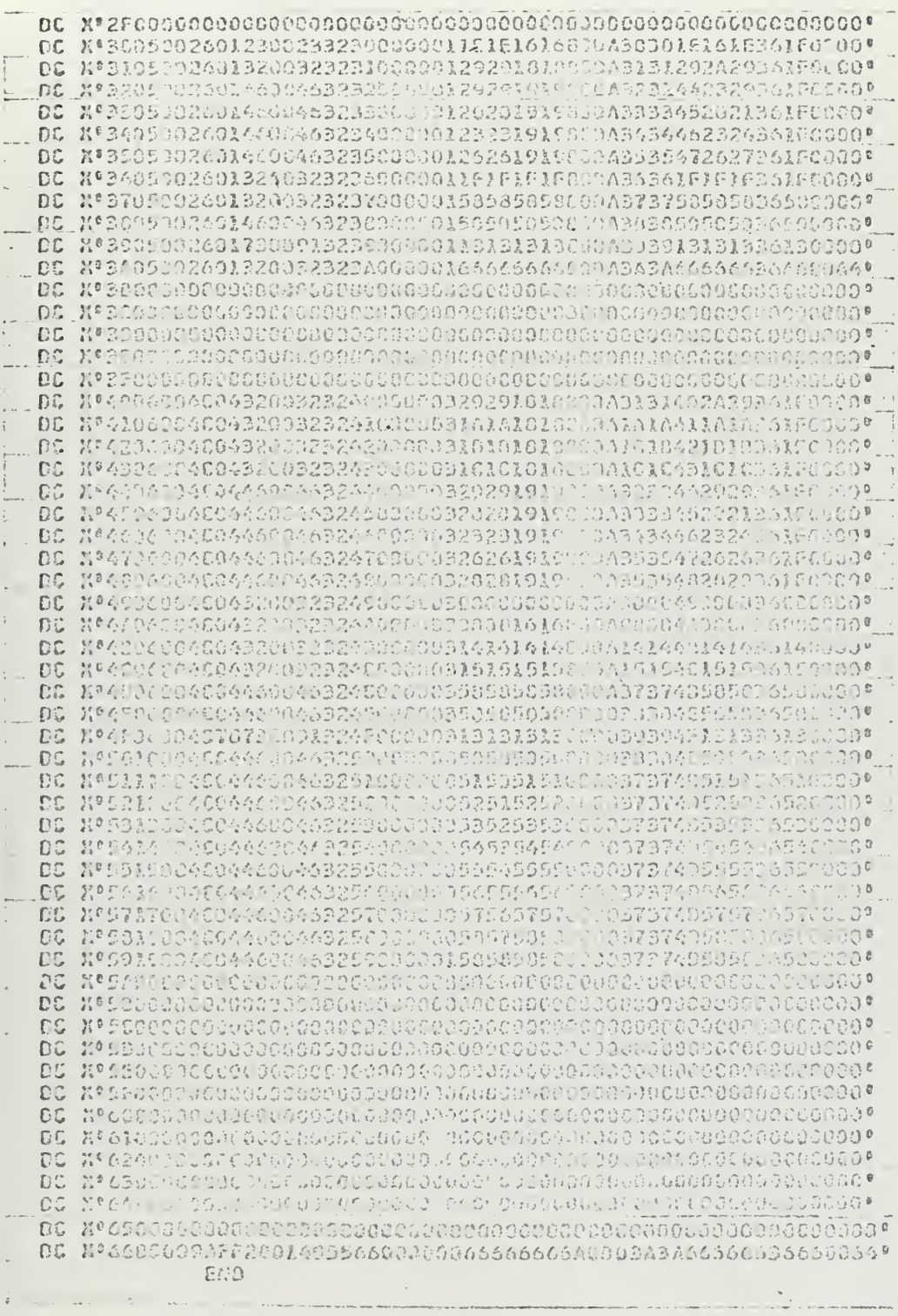

Figure 14 B1.(CONTINUED)

```
X'0005004C04200020640000000003F0000000A16162300000000000
X'0115002001100010640100000012A513E08800A17172441F081F1F0000
X'0215002001100010640200000012A513E08800A17172441F081F1F0000
X'0315002001100010640300000012A513E08800A17172441F081F1F0000
X'0415002001200020640400000012A513E08800A17172441F081F1F0000
X'0515002001200020640500000012A513E08800A17172441F081F1F0000
X'0615002001200020640600000012A513E08800A17172441F081F1F0000
X'0715002001100010640700000012A513E08800A17172441F071F1F0000
X'0815002001180018640800000012A513E08800A17172441F071F1F0000
X'0915002001200020640900000012A513E08800A17172441F081F1F0000
X'0A04004C04200020640A0000000A400A0A00011A1A220A0A0A0A0000
X'0B1D004C04100010640B012FFE3735390F800A1E1E270E0E0E0B0000
X'0C1D004C04100010640C012FFE3735390C800A1F1E270C0C0C0C0000
X'0D1D004C04100010640D012FFE3735390D800A1F1E270D0D0D0D0000
X'0E1D004C04100010640E012FFE3735390E800A1E1E270E0E0E0E0000
X'0F1D004C04100010640F012FFE3735390E800A1E1E270F0F0F0F0000
X'101D004C04100010641012FFE3735391080 0A1D1E271010101C0000
X'111D004C04100010641112FFE3735391180 0A1D1E271111111110000
X'121D004C04100010641212FFE3735391280 0A1F1E271212121220000
X'131D004C04100010641312FFE3735391380 0A1E1E271313131330000
X'140300 4C04200020641400000014141414003A1D1D211414141410000
X'150400 4C04200020641500000015151515800A1E1E201515151500000
X'161500 4C04200020641601 2F0100001600800A1616230000000000000
X'171500 4C04200020641701 2F012F303E08800A1818241F291F1F0000
X'181500 4C04300030641801 2F0130314308800A1919251F2F1F1F0000
X'191500 4C04500050641901 2F0131314408800A1919261F301F1F0000
X'1A1500 4C04200030641A012F010A0A0A0A800A1A1A220A0A1A1A0000
X'1B1500 4C04200020641B012F01393A1B0F800A1C1C27 4C4C1F1F0000
X'1C1500 4C04500050641C012F013D3F1C13800A1C1C28353513C10000
X'1D1500 4C04200030641D012F011414141480 0A1D1D211414141410000
X'1E1500 4C04200030641E012F011515151580 0A1E1E201515151500000
X'1F1400 4C04100010641F00000002A5101080 00A1717241F1F1F1F0000
X'201500 4C04200030642000001715151515800A1E1E201515151500000
X'211500 4C04200030642100001714141414800A1D1D211414141410000
X'221500 4C04200030642200001700A0A0A0A800A1A1A220A0A0A0A0000
X'231500 4C04200030642300001700000000800A1616230000000000000
X'241500 4C04100010642400001720D2F3E08800A1717241F291F1F0000
X'251500 4C04200030642500001720E2C4308800A1818251F2B1F1F0000
X'261500 4C04500050642600001731314408800A1919261F301F1F0000
X'271500 4C04100010642701 2F1737352708800A1E1E273333C3C0C0000
X'281500 4C04500050642801 2F173F3E2813800A1C1C2833331313130000
X'291500 4C0810001064290000002A2F3E08000A1717241F2F1F1F0000
```

Figure 14 B2. NEW IBM Research Schedule Table

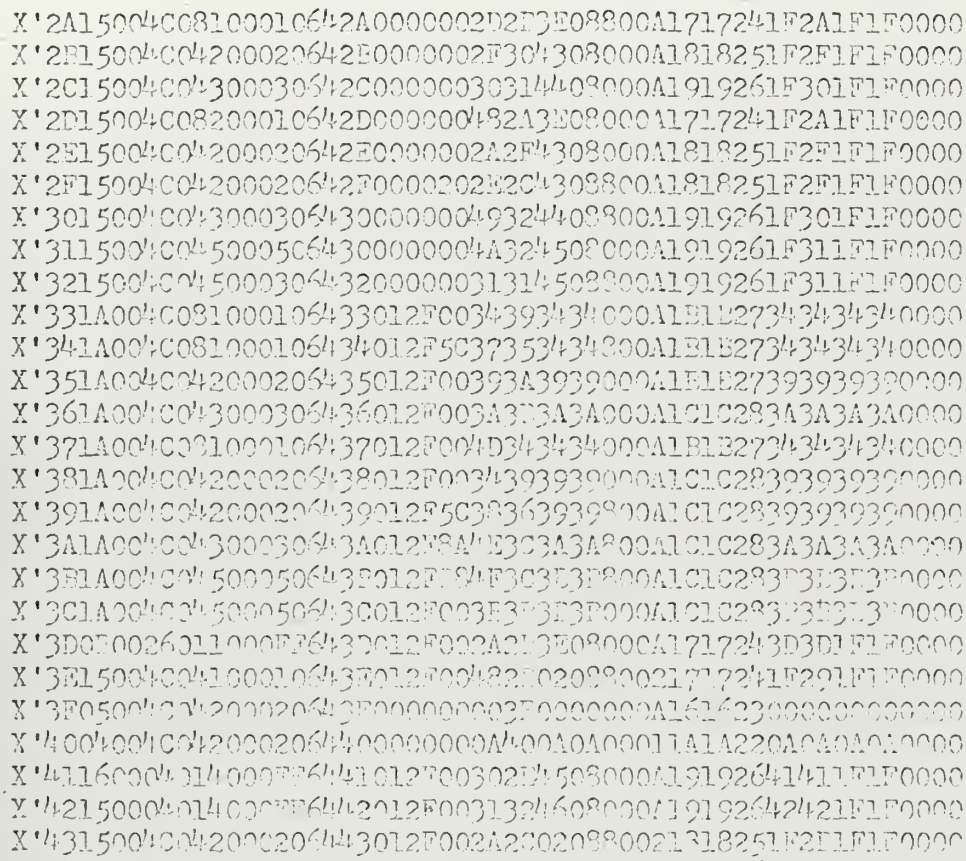

Figure 14 B2. (CONTINUED)

		LEVEL	PRI	TSVAL	QUANT	MAXCR	ST	ANTEX	DELTA	RPCRMMPPT	MRQ
	GROWING	29	15	004C	08	10	64	0000	0C	40	0A
	DELAYING	2A	15	0020	04	10	64	0000	06	80	0A
LOOPING	GROWING	2B	15	004C	04	20	64	0000	00	00	0A
INTER-	GROWING	2C	15	0020	04	30	64	0010	10	A0	0A
ACTIVE	GROWING	2D	16	004C	02	0C	64	0000	0C	40	0A
SET	GROWING	2E	16	004C	02	20	64	0010	10	A0	0A
	DELAYING	2F	15	004C	02	20	64	0000	10	80	0A
	DELAYING	30	15	004C	04	30	40	0020	24	A0	0A
	DELAYING	31	15	004C	04	50	50	0098	30	A0	2A
	DELAYING	32	16	004C	08	50	5A	0098	60	A0	46
LOOPING	SHRINKING	47	16	004C	10	08	64	0000	0C	C0	0A
INTER-	DELAYING	48	15	004C	08	08	64	0000	17	C0	0A
ACTIVE	SHRINKING	49	15	004C	02	30	40	0010	10	20	0A
SET	SHRINKING	4A	15	004C	04	40	50	004C	24	20	0A
	SHRINKING	4B	15	0026	08	15	64	0000	00	20	14
STARTING	INTERACTIVE	51	15	0020	02	20	64	0020	04	00	0A
SET	INTERACTIVE	52	14	0020	04	30	46	0010	08	A0	18
	INTERACTIVE	53	15	E3C1	C2	03	F2	0000	00	00	0A

Figure 14 B3. Test 4 (Run 5) Schedule Table Modifications

		LEVEL	QUANT	DELTA
	GROWING	29	08	0C
	DELAYING	2A	04	FE
	GROWING	2B	04	00
	GROWING	2C	02	10
LOOPING	GROWING	2D	01	0C
INTER-	GROWING	2E	01	10
ACTIVE	DELAYING	2F	02	FF
SET	DELAYING	30	02	FF
	DELAYING	31	02	FF
	DELAYING	32	02	FF
	SHRINKING	47	10	0C
LOOPING	DELAYING	48	08	FF
INTER-	SHRINKING	49	02	10
ACTIVE	SHRINKING	4A	02	24
SET	SHRINKING	4B	08	00
STARTING	INTERACTIVE	51	02	04
SET	INTERACTIVE	52	02	08
	INTERACTIVE	53	02	00

Figure 14 B4. Test 5 - Schedule Table Modifications

LIST OF REFERENCES

1. McCredie, J.N., Schlesinger, S.J., A Modular Simulation of TSS/360, paper presented at the Conference on Applications of Simulation, New York, New York, 9-11 December 1970.

2. Doherty, W.J., Scheduling TSS/360 for Responsiveness, Fall Joint Computer Conference, pp. 97-111, 1970.

3. Calingaert, P.,"System Performance Evaluation: Survey and Appraisal",Communications of the ACM, Vol. 10, No. 1, pp. 12-18, January 1967.

4. Hellerman, H.,"Some Principles of Time-Sharing Scheduler Strategies,"IBM Systems Journal, Vol. 8, No. 2, pp. 94-117, 1969.

5. Sayre, D., "Is Automatic 'Holding' of Programs Efficient Enough to Displace Manual?," CACM, Vol. 12, No. 12, pp. 656-660, 1969.

6. McCredie, J.N., Measurement Criteria for Virtual Memory Paying Rules, ACM proceedings, pp. 193-197, 1969.

7. Denning, P.J., "Thrashing, Its Causes and Prevention," Proceeding FJCC, Vol. 33, Part 1, pp. 915-922, 1968.

8. Brawn, B., Gustavson, F.G., Program Behavior in a Paying Environment, RC 2194, IBM Thomas J. Watson Research, York Town Heights, New York, 1968.

9. Technical Report Number 81, Virtual Memory, by P.J. Denning, 1970.

10. Denning, P.J., "The Working Set Model for Program Behavior," CACM, Vol. 11, No. 5, 1968.

11. Fine, G.H., Jackson, C., McIssac, P., Dynamic Program Behavior under Paging, Procedure ACM 21st National Conference, pp. 223-228, 1966.

12. IBM Corporation, Form Y28-2012,"System/360 Time Sharing System Resident Supervisor",Program Logic Manual, 1970.

13. IBM Corporation, Form Y28-2011, "System/360 Time Sharing System, System Control Blocks," Program Logic Manual, 1970.

14. System Development Corporation, Santa Monica, California, SP-3347, Benchmark Analysis of Time-Sharing Systems, by A.D. Karush, pp. 1-38, 30 June 1969.

15. Syms, G., Haines, W., Porterfield, J., "A Comparison of CP/67 and TSS/360 Time-Sharing System, U.S. Naval Postgraduate School, Monterey, California,"paper to be presented to the 3rd Symposium on Operating System Principles, Palo Alto, California, October 1971.

16. Stanford Computation Center, Stanford, California, The Analysis of General Purpose Computer Time-Sharing Systems, Doc. 40-10-1, by N.R. Nielsen, pp. 97-115, 1966.

17. Deniston, W.R., SIPE: A TSS/360 Software Measurement Technique, 24th National ACM Conference Proceedings, pp. 229-239, 1969.

18. De Meis, W.M., Weizer, N. , Measurement and Analysis of a Demand Paging Time-Sharing System, Proceeding of the FJCC, pp. 201-216, 1969.

19. Shulman, F.D.,"Hardware Measurement Device for IBM System/360 Time Sharing Evaluation,"Proceedings of the 22nd National Conference ACM, Vol. P-67, pp. 103-109, 1967.

20. Technical Report, TR53.0012, An Instruction-Trace Technique for Time-Sharing System/360, by C.E. Seabold, 31 March 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor Gordon H. Syms, Code 53 Zz Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Asst Professor George E. Heidorn, Code 55 Hd Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	Capt Jerry K. Baird, USMC 24122 Ramada Lane Mission Viejo, California 92675	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

Improving TSS/360 Performance by Tuning the Table-Driven Scheduler

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; June 1971

5. AUTHOR(S) (First name, middle initial, last name)

Jerry K. Baird

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1971	64	20

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

During the period of time from August 1970 through January 1971 and while employing the TSS/360 Time-Sharing System at this institution, it was observed by the user community that the performance of the system was poor compared to the previously used time-sharing system - the CP/67 (version 3, from Cambridge Research Center). For this reason, the problem of improving TSS/360 performance was undertaken as a thesis project. Specifically, the improvements consist of an increase in system performance - responsiveness and throughput - by judiciously adjusting the parameters of the TSS/360 Table-Driven Scheduler in accordance with the Principles of Balanced-Core Time and Working Set Size.

A number of test runs were made, and the results are given, employing different schedule tables. A set of benchmark programs (or script) were developed and used with these tests that were characteristic of a "typical" or "realistic" load at this installation.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
TSS/360						
Performance						
Time-Sharing Systems						
CP/67						

FORM 1 NOV 65 1473 (BACK)

01-807-6821

Baird

Improving TSS/360 performance by tuning the table-driven scheduler.

128136

Thesis
B149
c.1

Baird

Improving TSS/360 performance by tuning the table-driven scheduler.